CONTENTS

RELATED

How To Install nginx on CentOS 6 with yum

View

How To Install Linux, Apache, MySQL, PHP (LAMP) stack on Ubuntu

View

// Tutorial //

How To Deploy an ASP.NET Core Application with MySQL Server Using Nginx on Ubuntu 18.04

Published on July 23, 2019 · Updated on July 23, 2019

MySQL Deployment Applications Nginx Ubuntu 18.04

By Oluyemi Olususi
Developer and author at DigitalOcean.

The author selected the Open Source Initiative to receive a donation as part of the Write for DOnations program.

Introduction

ASP.NET Core is a high-performant, open-source framework for building modern web applications, meant to be a more modular version of Microsoft's ASP.NET Framework. Released in 2016, it can run on several operating systems such as Linux and macOS. This enables developers to target a particular operating system for development based on design requirements. With ASP.NET Core, a developer can build any kind of web application or service irrespective of the complexity and size. Developers can also make use of Razor pages to create page-focused design working on top of the traditional Model-View-Controller (MVC) pattern.

ASP.NET Core provides the flexibility to integrate with any front-end frameworks to handle client-side logic or consume a web service. You could, for example, build a RESTful API with ASP.NET Core and easily consume it with JavaScript frameworks such as Angular, React, and Vue.js.

In this tutorial you'll set up and deploy a production-ready ASP.NET Core application with a MySQL Server on Ubuntu 18.04 using Nginx. You will deploy a demo ASP.NET Core application similar to the application from Microsoft's documentation ar

GitHub. Once deployed, the demo application will allow you to create a list of movies and store it in the database. You'll be able to create, read, update, and delete records from the database. You can use this tutorial to deploy your own ASP.NET Core application instead; it's possible you'll have to implement extra steps that include generating a new migration file for your database.

Prerequisites

You will need the following for this tutorial:

- One Ubuntu 18.04 server set up by following the Ubuntu 18.04 initial server setup guide, including a non-root user with `sudo` access and a firewall.
- Nginx installed by following How To Install Nginx on Ubuntu 18.04.
- A secured Nginx web server. You can follow this tutorial on How To Secure Nginx with Let's Encrypt on Ubuntu 18.04.
- Both of the following DNS records set up for your server. You can follow this introduction to DigitalOcean DNS for details on how to add them.
    - An `A` record with `your-domain` pointing to your server's public IP address.
    - An `A` record with `www.your-domain` pointing to your server's public IP address.

- MySQL installed by following How To Install the Latest MySQL on Ubuntu 18.04.

Step 1 – Installing .NET Core Runtime

A .NET Core runtime is required to successfully run a .NET Core application, so you'll start by installing this to your machine. First, you need to register the Microsoft Key and product repository. After that, you will install the required dependencies.

First, logged in as your new created user, make sure you're in your root directory:

```
$ cd ~
```
Copy

Next, run the following command to register the Microsoft key and product repository:

```
$ wget -q https://packages.microsoft.com/config/ubuntu/18.04/packages-microsc
```
Copy
```
d.
```

Use `dpkg` with the `-i` flag to install the specified file:

```
$ sudo dpkg -i packages-microsoft-prod.deb
```
Copy

To facilitate the installation of other packages required for your application, you will install the `universe` repository with the following command:

```
$ sudo add-apt-repository universe
```
Copy

Next install the `apt-transport` package to allow the use of repositories accessed via the HTTP Secure protocol:

```
$ sudo apt install apt-transport-https
```
Copy

Now, run the following command to download the packages list from the repositories and update them to get information on the newest versions of packages and their dependencies:

```
$ sudo apt update
```
Copy

Finally, you can install the .NET runtime SDK with:

```
$ sudo apt install dotnet-sdk-2.2
```
Copy

You will be prompted with the details of the size of additional files that will be installed. Type `Y` and hit `ENTER` to continue.

Now that you're done installing the .NET Core runtime SDK on the server, you are almost ready to download the demo application from GitHub and set up the deployment configuration. But first, you'll create the database for the application.

Step 2 – Creating a MySQL User and Database

In this section, you will create a MySQL server user, create a database for the application, and grant all the necessary privileges for the new user to connect to the database from your application.

To begin, you need to access the MySQL client using the MySQL root account as shown here:

```
$ mysql -u root -p
```
Copy

You will be prompted to enter the root account password, set up during the prerequisite tutorial.

Next, create a MySQL database for the application with:

```
mysql> CREATE DATABASE MovieAppDb;
```
Copy

You will see the following output in the console:

```
Output
Query OK, 1 row affected (0.03 sec)
```

You've now created the database successfully. Next, you will create a new MySQL user, associate them with the newly created database, and grant them all privileges.

Run the following command to create the MySQL user and password. Remember to change the username and password to something more secure:

```
mysql> CREATE USER 'movie-admin'@'localhost' IDENTIFIED BY 'password';
```
Copy

You will see the following output:

```
Output
Query OK, 0 rows affected (0.02 sec)
```

To access a database or carry out a specific action on it, a MySQL user needs the appropriate permission. At the moment **movie-admin** does not have the appropriate permission over the application database.

You will change that by running the following command to grant access to **movie-admin** on `MovieAppDb`:

```
mysql> GRANT ALL PRIVILEGES ON MovieAppDb.* TO 'movie-admin'@'localhost';
```
Copy

You will see the following output:

Output

```
Query OK, 0 rows affected (0.01 sec)
```

Now, you can reload the grant tables by running the following command to apply the changes that you just made using the flush statement:

```
mysql> FLUSH PRIVILEGES;
```
Copy

You will see the following output:

Output

```
Query OK, 0 rows affected (0.00 sec)
```

You are done creating a new user and granting privileges. To test if you are on track, exit the MySQL client:

```
mysql> quit;
```
Copy

Log in again, using the credentials of the MySQL user you just created and enter the appropriate password when prompted:

```
$ mysql -u movie-admin -p
```
Copy

Check to be sure that the user **movie-admin** can access the created database, check with:

```
mysql> SHOW DATABASES;
```
Copy

You will see the `MovieAppDb` table listed in the output:

```
Output
+--------------------+
| Database           |
+--------------------+
| MovieAppDb         |
| information_schema |
+--------------------+
2 rows in set (0.01 sec)
```

Now, exit the MySQL client:

```
mysql> quit;
```
Copy

You've created a database, made a new MySQL user for the demo application, and granted the newly created user the right privileges to access the database. In the next section, you will start setting up the demo application.

Step 3 – Setting Up the Demo App and Database Credentials

As stated earlier, you'll deploy an existing ASP.NET Core application. This application was built to create a movie list and it uses the Model-View-Controller design pattern to ensure a proper structure and separation of concerns. To create or add a new movie to the list, the user will populate the form fields with the appropriate details and click on the **Create** button to post the details to the controller. The controller at this point will receive a POST HTTP request with the submitted details and persist the data in the database through the model.

You will use Git to pull the source code of this demo application from GitHub and save it in a new directory. You could also download an alternate application here if you will be deploying a different application.

To begin, create a new directory named `movie-app` from the terminal by using the following command:

```
$ sudo mkdir -p /var/www/movie-app
```
Copy

This will serve as the root directory for your application. Next, change the folder owner and group in order to allow a non-root user account to work with the pr

```
$ sudo chown sammy:sammy /var/www/movie-app
```
Copy

Replace **sammy** with your sudo non-root username.

Now, you can move into the parent directory and clone the application on GitHub:

```
$ cd /var/www
$ git clone https://github.com/do-community/movie-app-list.git movie-app
```
Copy

You will see the following output:

```
Output
Cloning into 'movie-app'...
remote: Enumerating objects: 91, done.
remote: Counting objects: 100% (91/91), done.
remote: Compressing objects: 100% (73/73), done.
remote: Total 91 (delta 13), reused 91 (delta 13), pack-reused 0
Unpacking objects: 100% (91/91), done.
```

You have successfully cloned the demo application from GitHub, so the next step will be to create a successful connection to the application database. You will do this by editing the `ConnectionStrings` property within the `appsettings.json` file and add the details of the database.

Change directory into the application:

```
$ cd movie-app
```
Copy

Now open the file for editing:

```
$ sudo nano appsettings.json
```
Copy

Add your database credentials:

appsettings.json

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Warning"
```

```
    }
  },
  "AllowedHosts": "*",
  "ConnectionStrings": {
    "MovieContext": "Server=localhost;User Id=movie-admin;Password=password;Database=
  }
}
```

With this in place, you've successfully created a connection to your database. Now press `CTRL+X` to save your changes to the file and type `Y` to confirm. Then hit `ENTER` to exit the page.

ASP.NET Core applications use a .NET standard library named Entity Framework (EF) Core to manage interaction with the database. Entity Framework Core is a lightweight, cross-platform version of the popular Entity Framework data access technology. It is an object-relational mapper (ORM) that enables .NET developers to work with a database using any of the database providers, such as MySQL.

You can now update your database with the tables from the cloned demo application. Run the following command for that purpose:

```
$ dotnet ef database update
```

Copy

This will apply an update to the database and create the appropriate schemas.

Now, to build the project and all its dependencies, run the following command:

```
$ dotnet build
```

Copy

You will see output similar to:

```
Output
Microsoft (R) Build Engine version 16.1.76+g14b0a930a7 for .NET Core
Copyright (C) Microsoft Corporation. All rights reserved.

  Restore completed in 95.09 ms for /var/www/movie-app/MvcMovie.csproj.
  MvcMovie -> /var/www/movie-app/bin/Debug/netcoreapp2.2/MvcMovie.dll
  MvcMovie -> /var/www/movie-app/bin/Debug/netcoreapp2.2/MvcMovie.Views.dll

Build succeeded.
    0 Warning(s)
    0 Error(s)
```

```
Time Elapsed 00:00:01.91
```

This will build the project and install any third-party dependencies listed in the `project.assets.json` file but the application won't be ready for production yet. To get the application ready for deployment, run the following command:

```
$ dotnet publish
```
Copy

You will see the following:

```
Output
Microsoft (R) Build Engine version 16.1.76+g14b0a930a7 for .NET Core
Copyright (C) Microsoft Corporation. All rights reserved.

Restore completed in 89.62 ms for /var/www/movie-app/MvcMovie.csproj.
MvcMovie -> /var/www/movie-app/bin/Debug/netcoreapp2.2/MvcMovie.dll
MvcMovie -> /var/www/movie-app/bin/Debug/netcoreapp2.2/MvcMovie.Views.dll
MvcMovie -> /var/www/movie-app/bin/Debug/netcoreapp2.2/publish/
```

This will pack and compile the application, read through its dependencies, publish the resulting set of files into a folder for deployment, and produce a cross-platform `.dll` file that uses the installed .NET Core runtime to run the application.

By installing dependencies, creating a connection to the database, updating the database with the necessary tables, and publishing it for production, you've completed the setup for this demo application. In the next step you will configure the web server to make the application accessible and secure at your domain.

Step 4 – Configuring the Web Server

By now, having followed the How To Secure Nginx with Let's Encrypt tutorial, you'll have a server block for your domain at `/etc/nginx/sites-available/your_domain` with the `server_name` directive already set appropriately. In this step, you will edit this server block to configure Nginx as a reverse proxy for your application. A reverse proxy is a server that sits in front of web servers and forwards every web browser's request to those web servers. It receives all requests from the network and forwards them to a different web server.

In the case of an ASP.NET Core application, Kestrel is the preferred web server that is included with it by default. It is great for serving dynamic content from

application as it provides better request-processing performance and was designed to make ASP.NET as fast as possible. However, Kestrel isn't considered a full-featured web server because it can't manage security and serve static files, which is why it is advisable to always run it behind a web server.

To begin, ensure that you are within the root directory of your server:

```
$ cd ~
```
Copy

Open the server block for editing with:

```
$ sudo nano /etc/nginx/sites-available/your_domain
```
Copy

As detailed in the Step 4 of the How To Secure Nginx with Let's Encrypt tutorial, if you selected option 2, Certbot will automatically configure this server block in order to redirect HTTP traffic to HTTPS with just a few modifications.

Continue with the configuration by editing the first two blocks in the file to reflect the following:

/etc/nginx/sites-available/your-domain

```
server {

    server_name your-domain  www.your-domain;

   location / {
         proxy_pass http://localhost:5000;
         proxy_http_version 1.1;
         proxy_set_header Upgrade $http_upgrade;
         proxy_set_header Connection keep-alive;
         proxy_set_header Host $host;
         proxy_cache_bypass $http_upgrade;
         proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
         proxy_set_header X-Forwarded-Proto $scheme;
        }

listen [::]:443 ssl ipv6only=on; # managed by Certbot
listen 443 ssl; # managed by Certbot
ssl_certificate /etc/letsencrypt/live/your-domain/fullchain.pem; # managed by Certbot
ssl_certificate_key /etc/letsencrypt/live/your-domain/privkey.pem; # managed by Certb
include /etc/letsencrypt/options-ssl-nginx.conf; # managed by Certbot
ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem; # managed by Certbot
```

```
}
...
```

The configuration in this server block will instruct Nginx to listen on port `443`, which is the standard port for websites that use SSL. Furthermore, Nginx will accept public traffic on port `443` and forward every matching request to the built-in Kestrel server at `http://localhost:5000`.

Finally, following the server block you just edited in the file, ensure that the second server block looks like so:

/etc/nginx/sites-available/your-domain

```
...
server {
if ($host = www.your-domain) {
    return 301 https://$host$request_uri;
} # managed by Certbot


if ($host = your-domain) {
    return 301 https://$host$request_uri;
} # managed by Certbot


    listen 80;
    listen [::]:80;

    server_name your-domain  www.your-domain;
return 404; # managed by Certbot
}
```

This server block will redirect all requests to `https://your-domain` and `https://www.your-domain` to a secure HTTPS access.

Next, force Nginx to pick up the changes you've made to the server block by running:

```
$ sudo nginx -s reload
```

Copy

With the Nginx configuration successfully completed, the server is fully set up to forward all HTTPS requests made to `https://your-domain` on to the ASP.NET Core app running on Kestrel at `http://localhost:5000`. However, Nginx isn't set up to man

server process. To handle this and ensure that the Kestrel process keeps running in the background, you will use `systemd` functionalities.

Systemd files will allow you to manage a process by providing start, stop, restart, and log functionalities once you create a process of work called a unit.

Move into the `systemd` directory:

```
$ cd /etc/systemd/systems
```
Copy

Create a new file for editing:

```
$ sudo nano movie.service
```
Copy

Add the following content to it:

movie.service

```
[Unit]
Description=Movie app

[Service]
WorkingDirectory=/var/www/movie-app
ExecStart=/usr/bin/dotnet /var/www/movie-app/bin/Debug/netcoreapp2.2/publish/MvcMovie
Restart=always
RestartSec=10
SyslogIdentifier=movie
User=sammy
Environment=ASPNETCORE_ENVIRONMENT=Production
Environment=DOTNET_PRINT_TELEMETRY_MESSAGE=false

[Install]
WantedBy=multi-user.target
```

The configuration file specifies the location of the project's folder with `WorkingDirectory` and the command to execute at the start of the process in `ExecStart`. In addition, you've used the `RestartSec` directive to specify when to restart the `systemd` service if the .NET runtime service crashes.

Now save the file and enable the new movie service created with:

```
$ sudo systemctl enable movie.service
```
Copy

After that, proceed to start the service and verify that it's running by starting the service:

```
$ sudo systemctl start movie.service
```
Copy

Then check its status:

```
$ sudo systemctl status movie.service
```
Copy

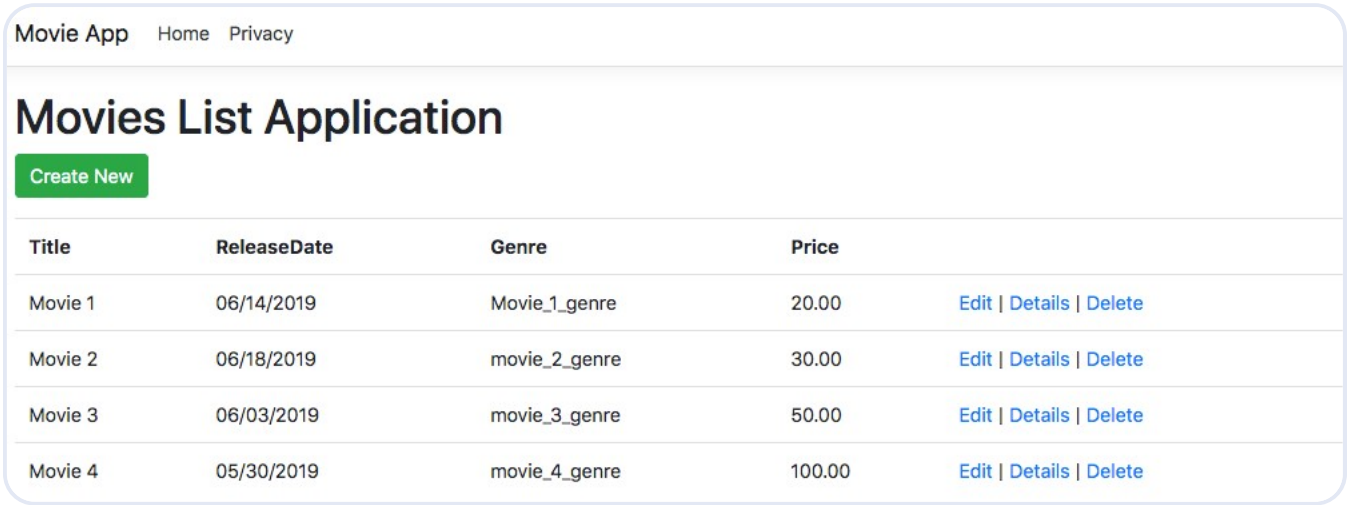
You will see the following output:

```
Output
movie.service - Movie app
   Loaded: loaded (/etc/systemd/system/movie.service; enabled; vendor preset: enabled
   Active: active (running) since Sun 2019-06-23 04:51:28 UTC; 11s ago
 Main PID: 6038 (dotnet)
    Tasks: 16 (limit: 1152)
   CGroup: /system.slice/movie.service
           └─6038 /usr/bin/dotnet /var/www/movie-app/bin/Debug/netcoreapp2.2/publish/
```

This output gives you an overview of the current status of the `movie.service` created to keep your app running. It indicates that the service is enabled and currently active.

Navigate to `https://your-domain` from your browser to run and test out the application.

You'll see the home page for the demo application—**Movie List Application**.



With the reverse proxy configured and Kestrel managed through systemd, the web app is fully configured and can be accessed from a browser.

Conclusion

In this tutorial, you deployed an ASP.NET Core application to an Ubuntu server. To persist and manage data, you installed and used MySQL server and used the Nginx web server as a reverse proxy to serve your application.

Beyond this tutorial, if you're interested in building an interactive web application using C# instead of Javascript you could try a web UI framework by Microsoft called Blazor. It is an event-driven component-based web UI for implementing logic on the client side of an ASP.NET Core application.

If you wish to deploy your own application, you'll need to consider other required procedures to deploy your app. The complete source code for this demo application can be found here on GitHub.

About the authors

Oluyemi Olususi Author

Developer and author at DigitalOcean.

Kathryn Hancox Editor

Developer and author at DigitalOcean.

Still looking for an answer?

Ask a question | Search for more help

Was this helpful? Yes No

Comments

10 Comments

Leave a comment...

This textbox defaults to using `Markdown` to format your answer.

You can type `!ref` in this text area to quickly search our full set of tutorials, documentation & marketplace offerings and insert the link!

Sign In or Sign Up to Comment

dougan778 • February 5, 2021

I was able to get this tutorial to work with Core 3.1 with a few minor updates. Here are the changes (mostly aggregating feedback from other comments):

1. switch sdk install to: sudo apt install dotnet-sdk-3.1
2. Core 3.1 split out an EF package from the SDK, so you need to install it manually, since it isn't there by default anymore: dotnet tool install –global dotnet-ef
3. Fix a port 5001 timeout issue. Forwarded headers, from https://docs.microsoft.com/en-us/aspnet/core/host-and-deploy/linux-nginx?view=aspnetcore-3.1, to quote mikeg70:

From the Microsoft documentation above, the only new edit you will need to apply is in the Startup.cs file. Add the following code in your Configure method

using Microsoft.AspNetCore.HttpOverrides; …

app.UseForwardedHeaders(new ForwardedHeadersOptions { ForwardedHeaders = ForwardedHeaders.XForwardedFor | ForwardedHeaders.XForwardedProto });

Add the following nuget package to your .NET project: Microsoft.AspNetCore.HttpOverrides –version 2.2.0 through VS UI or command line: dotnet add package Microsoft.AspNetCore.HttpOverrides –version 2.2.0

4. Be aware of typo, /etc/systemd/systems should be "system" in original tutorial

My application is using Postgres so I didn't test the MySQL stuff, but I wouldn't suspect that has changed.

Reply

Bigpop • January 7, 2021

As the date of of today this tutorial is broken. As dotnet 2.1 is not supported anymore and .net 5 is the new way to go. It still usefull but I didn't

install up to the point of creating the db with entity framework the entity framework part is broken the rest still works. Plus I didn't manage to make the entity framework dotnet ef migration works on Ubuntu server 20.04 LTS still Investagating how I can resolve my issues. As I need to deploy my application on a Linux production server If I'm losing too much time configuring I may consider switching technologies all together as python and django are more straight forward regarding set up on Linux... I like the .net framework but if it is still a configuration hell with broken part and dependency hell... I might ass well wait until it is ready and mature technology.

Show replies Reply

JuanG4 • February 14, 2020

Note: On your deployed application you do NOT want to have any passwords or user secrets in your source code, and certainly not on version control (git)

Reply

307a8be1cc304e198a6c47aa72 • August 9, 2022

I don't get here just one thing. How did you point domain to the application? Should I move dlls into my domain folder?

Reply

Xpyder • January 13, 2022

In case anyone else runs into this...

before running `dotnet ef database update`

I had to run

```
dotnet tool install --global dotnet-ef
```

```
dotnet tool update --global dotnet-ef
```

I'm using dotnet 6 as listed here: https://docs.microsoft.com/en-us/dotnet/core/install/linux-ubuntu

Reply

oliverguy6 • April 8, 2021

I am getting the error `main process exited code=dumped status=6/ABRT` when I do `sudo systemctl status movie.service` I haven't got a clue why, any solutions?

Show replies Reply

jtech • December 23, 2020

I added following under site-available file. Prior to adding this, got 404 for js, css, images

location ~* /(css|js|lib|fonts|media) { root /var/www/your-app/html/wwwroot; }

Show replies Reply

ianEsq • November 24, 2020

You need to make a small correction to the instruction to edit the service. It should be:

```
cd /etc/systemd/system
```

You have it as:

```
cd /etc/systemd/systems
```

Reply

ypilevneDolphin • July 26, 2020

cd /etc/systemd/systems should be cd /etc/systemd/system I think.

Reply

viyankatech • April 2, 2020

There is a typo. While navigating to systemsd step, it is given as

cd /etc/systemd/systems

But it is,

cd /etc/systemd/system

Its not systems, but system. Spent 0.5 hrs wondering why I could not find that folder :(

Reply

Load More Comments

Popular Topics

Ubuntu

Linux Basics

JavaScript

React

Python

Security

MySQL

Docker

Kubernetes

Browse all topic tags

Free Managed Hosting →

All tutorials →

Questions

Q&A Forum

Ask a question

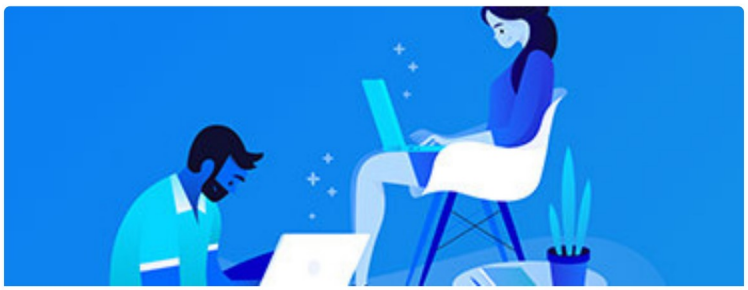
DigitalOcean Support

GET OUR BIWEEKLY NEWSLETTER

Sign up for Infrastructure as a Newsletter.

HOLLIE'S HUB FOR GOOD

Working on improving health and education, reducing inequality, and spurring economic growth? We'd like to help.



BECOME A CONTRIBUTOR